CSCI 6907.11

Adv. Net. Sys. Prog.

Lecture 12 - Network Programming Paradigms

Tim Wood
CS@GWU
2015

Today

Warmup: Sockets and Data

Multi-threading

Non-blocking I/O

Event-driven programming

Socket Data

Get today's code:

- Make a fork of **gwAdvNet2015/lec-12-code** and clone it
- run: **`git checkout simple-messager`**

Take a look at the code in messager/

- Nothing should be too surprising
- Note that messy socket stuff has been moved to a header file
- Always try to keep your code looking clean!

Socket Data

Write an “America Converter” server

- Clients send a temperature and a length
- Server converts temperature from F to C
- Server converts length from feet and inches to meters
- Client should use **`scanf()`** to read in data to be sent
- Server prints to screen and sends back converted values
- Use **messager** as a base

1 foot = 12 inches
1 meter = 39.370 inches
C = (F-32) * 5/9

If you finish earlier, check the Bug in this repo’s Issues

Application Layer Protocols

This is why we need protocols like HTTP

- Specify exactly what data to expect and in what order

Most application protocols don't follow a standard

- Just be sure your code or documentation explains the steps!

Sometimes an application supports multiple protocols

- Why?

Memcached K-V Store

Binary Protocol

https://code.google.com/p/memcached/wiki/MemcacheBinaryProtocol

```
Request header:

  Byte/     0       |       1       |       2       |       3       |
     /              |               |               |               |
    |0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|
    +---------------+---------------+---------------+---------------+
   0| Magic         | Opcode        | Key length                    |
    +---------------+---------------+---------------+---------------+
   4| Extras length | Data type     | Reserved                      |
    +---------------+---------------+---------------+---------------+
   8| Total body length                                             |
    +---------------+---------------+---------------+---------------+
  12| Opaque                                                        |
    +---------------+---------------+---------------+---------------+
  16| CAS                                                           |
    |                                                               |
    +---------------+---------------+---------------+---------------+
    Total 24 bytes
```

```
Byte/     0       |       1       |       2       |       3       |
   /              |               |               |               |
  |0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|
  +---------------+---------------+---------------+---------------+
  0/ HEADER                                                       /
   /                                                              /
   /                                                              /
   /                                                              /
   +---------------+---------------+---------------+---------------+
  24/ COMMAND-SPECIFIC EXTRAS (as needed)                         /
  +/  (note length in the extras length header field)             /
   +---------------+---------------+---------------+---------------+
  m/ Key (as needed)                                              /
  +/  (note length in key length header field)                    /
   +---------------+---------------+---------------+---------------+
  n/ Value (as needed)                                            /
  +/  (note length is total body length header field, minus       /
  +/   sum of the extras and key length body fields)              /
   +---------------+---------------+---------------+---------------+
   Total 24 bytes
```

ASCII Protocol

https://github.com/memcached/memcached/blob/master/doc/protocol.txt

- `<command name> <key> <flags> <exptime> <bytes> [noreply]\r\n`
- `GET somekey`

Network Apps Multi-Task

What happens when you call…?

- recv() or read()
- send() or write()
- listen()
- accept()
- bind()

Most servers must do multiple things at once:

- Handle many incoming requests simultaneously

How do we do this?

Threads + Blocking I/O

If a call is blocking, we can run it within a thread so that other threads can still do useful work!

```
int pthread_create(pthread_t *thread,
     const pthread_attr_t *attr,
     void *(*start_routine)(void *),
     void *arg)
```

thread ID

thread attributes often NULL

function to run in thread

argument to function

See a sample: https://github.com/gwAdvNet2015/adv-net-samples/blob/master/threads/server-tcp.c

man pthreads

Other important functions:

```
int pthread_join(pthread_t thread, void **value_ptr)
        Wait for the termination of the specified thread, then clear its state.

int pthread_detach(pthread_t thread)
       Marks a thread for deletion when its function ends.

pthread_t pthread_self(void)
        Returns the thread ID of the calling thread.


int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *attr)
        Initialize a mutex with specified attributes.

int pthread_mutex_lock(pthread_mutex_t *mutex)
        Lock a mutex and block until it becomes available.

int pthread_mutex_trylock(pthread_mutex_t *mutex)
        Try to lock a mutex, but don't block if the mutex is locked by another
        thread, including the current thread.

int pthread_mutex_unlock(pthread_mutex_t *mutex)
        Unlock a mutex.
```

Multi-Threaded Converter

What happens when two clients try to connect to your conversion server?

How can we make it support multiple clients?

- When do we start threads? What do they do?

Need help? Take a look at:

https://github.com/gwAdvNet2015/adv-net-samples/blob/master/threads/server-tcp.c

If you finish earlier, check the Bug in this repo's Issues

Multi-Threaded Chat

Let's look at messager/threaded-server-msg.c

Multi-Threaded Chat

Let's look at messager/threaded-server-msg.c

What if we wanted the server to send all messages it receives back to all connected clients?

Shared State

The source of all life's [performance] problems

Ensuring consistency is **hard** and **slow**

- Locks, condition variables, test and set, compare and swap

What if we could have a single thread do all the work, but not block all the time?

Non-Blocking I/O

Don't want to wait? Easy, use non-blocking sockets

```
#include <fcntl.h>
.
.
.
sockfd = socket(PF_INET, SOCK_STREAM, 0);
fcntl(sockfd, F_SETFL, O_NONBLOCK);
```

Once configured, any calls that normally would block will instead return -1

- accept, recv, read

Now it is your job to repeatedly check the socket until it has data ready!

Polling

Polling is the act of repeatedly checking something to see if it is ready

```
/* Set socket to be NON-Blocking */
fcntl(clientfd, F_SETFL, O_NONBLOCK);

while(1) {
        bytes_read = read(clientfd, message, sizeof message);
        if(bytes_read > 0) {
                /* The socket has data ready */
       }
        else if(bytes_read == 0) {
                /* Client closed socket */
                break;
        }
        else {
                /* No data is waiting on socket... */
                usleep(10000); /* Sleep for 10 millisec */
        }
}
```

Polling Chat Server

Look at **`nonb/nonb-server-msg.c`** on **`master`** branch

Does this use threads or polling? Does it use blocking or non-blocking I/O?

Is this efficient?

Why might you use this?

Polling is kind of terrible

What if we don't want to poll, but we do want to be able to check the status of several different sockets from a single thread?

We need a way to ask the OS to tell us when one of several different sockets has data

- We will block…
- but we will be notified by several different events!

Select

The **`select`** function is used for Event-driven I/O

- The benefits of non-blocking I/O, but more efficient

```
int select(int numfds, fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds, struct timeval *timeout);
```

Works with sets of file descriptors

- Sockets, files, stdin, etc

Handles events:

- read, write, exception

```
FD_SET(int fd, fd_set *set);
        Add fd to the set.

FD_CLR(int fd, fd_set *set);
        Remove fd from the set.

FD_ISSET(int fd, fd_set *set);
        Return true if fd is in the set.

FD_ZERO(fd_set *set);
        Clear all entries from the set.
```

Select-Based Chat

Look at **`nonb/select-server-msg.c`** on **`master`** branch

How does it use select?

How could this be improved?

- Do it!

How does this kind of programming compare to writing multi-threaded programs?

Event Driven Programming

Select is used in event-driven programming

- Can be much more efficient than multi-threaded programs.
- Why?

Multi-threaded programs

- Pros? Cons?

Event-driven programs

- Pros? Cons?

(of course you can also have multi-threaded programs that are event-driven)

Issue Triage

What will you complete before next class?

- What should we remove? What should we add?

Project proposals **due Sunday 4/5**

If you didn't submit to GENI come up with something!

- POX, Sockets, multi-threading, complex data structures, etc

If you did submit to GENI, congrats!

- You should try to fix the parts that don't work